CLAIMS:

What is claimed is:

1. A method using a computer which automatically generates and executes machine-code, comprising the steps of

   a) preventing the multi-tasking of the operation-system, by setting the interrupt-mask of the processor to NMI or using a multitasking-disable-routine of the operating-system;

   b) capturing the processors exception-vectors by own analysis-routines;

   c) generating normal numbers and writing them into memory;

   d) backing up the current values of the processors registers;

   e) positioning the instruction-pointer = program-counter to the generated number in memory and executing the number like it would be a processor-opcode; and

   f) analysing the effects of this opcode-like execution of the number and storing the analysis-results, p.e. in a database.

2. A method according to claim 1, wherein said step of (d) "backing up the current values of the processors registers" comprising the steps of:

   a) not only saving them for a later comparison but setting them to predefined initial conditions;

   b) setting them not only one time but several times to many different predefined initial conditions which means several executions of one number in step (e) by these different initial conditions for to have a more efficient analysis-determinations of possible number-execution-effects in step (f).

3. A method according to claim 2, wherein said step (1f) "analysing the effects of this opcode-like execution of the number" further comprising the step of determining the number=opcode's mnemonic and its related source- and destination-registers by regarding all execution-effects of every initial condition.

4. A method according to claim 3, wherein said step (1c) "generating normal numbers and writing them into memory" comprising the steps of combinations, which means:

a) taking one number which analysed results are already stored and appending another number with stored analysis-results to analyze the execution-effects of this two-number-combination and store the result.

b) combining 2-number-combinations, which effects are already analysed, with a further analysed number and analysing and storing the analysis-results of the effects of these 3-number-combinations.

c) combining a 3-number-combination, which effects are already analysed, with a further number, which effects are already analysed, or combining two 2-number-combinations, which effects are already analysed, and analysing and storing the effects of these 4-number-combinations.

d) combining larger combinations, which effects are already analysed, with numbers or combinations, which effects are already analysed, and analysing and storing the effects of these larger combinations.

5. A method according to claim 4, further comprising the step of using only combinations for further use, which got a positive value from a valuation-function, which appraises the valuability of the combination in reference to reaching a pregiven programming-aim, not causing fatal exceptions, not overwriting exception-vectors or the program, avoiding to use forbidden registers or extensive writes to memory or large jumps, etc.

6. A method according to claim 5, further comprising the step of valuating and changing the valuation-function of the dynamic valuation-system by a meta-valuation-function valuating the results of the valuation-function according to clustering to boundary-values, low-values, other fixed values, etc., and then revaluating the results of the new valuation-function.

7. A method according to claim 4, further comprising the step of implementing calls to operation-system routines which are offered in a table with entrance-address and source- and destination-registers.

8. A method according to claim 7, further comprising the step of using only calls and combinations for further use, which got a positive value from a valuation-function, which appraises the valuability of the call-combination in reference to reaching a pregiven programming-aim, not causing fatal exceptions, not overwriting exception-vectors or the program, avoiding to use forbidden registers or extensive writes to memory or large jumps, etc.

9. A method according to claim 8, further comprising the step of offering the disassembly of the solution-programs which solved the programming-aim.

10. A method using a computer which automatically generates and executes machine-code,

comprising the steps of

a) capturing the tasks=processes exception-vectors by own analysis-routines;

b) generating normal numbers and writing them into memory;

c) backing up the current values of the processors registers;

d) positioning the instruction-pointer=program-counter to the generated number in memory and executing the number like it would be a processor-opcode; and

e) analysing the effects of this opcode-like execution of the number and storing the analysis-results, p.e. in a database.

11. A method according to claim 10, further comprising the steps of modelling the following basic needs:

a) "no_pain", where pain means damage to the own program, which is an overwriting of the own machine-code, which is recognised by comparing the programs checksum after every execution of a number or number-combination, and repairing damaged parts of the own machine-code from a duplication, which causes a decrementation of the energy-register for every damaged opcode of the own program which now has to be copied for reparation; and

b) "no_hunger", where hunger means the imminent loss of energy, where energy is modelled by the value of a predefined register, which causes negative effects on low values like
- the loss of the capability of appraising combinations referable to the programming aim on low values of the energy-register,
- mistakes on the valuation of the combination-execution concerning the source-registers on very low values of the energy-register,
- the loss of the capability of self-repairing on "pain" on extreme low values of the energy-register,
- a hardware-dependant decreasing of the power supply of the RAM (p.e. by increasing a resistor) on two times in series extreme low values of the energy-register,
- a hardware-dependant decreasing of the power supply of the processor (p.e. by increasing a resistor) on three times in series extreme low values of the energy-register,

and this energy-register is decremented after every action, where action means the execution of a number.

12. A method according to claim 10, wherein said step of (10c) "backing up the current values of

the processors registers" further comprising the steps of

a) not only saving them for a later comparison but setting them to predefined initial conditions;

b) setting them not only one time but several times to many different predefined initial conditions which means several executions of one number in step (10d) by these different initial conditions for to have a more efficient analysis-determinations of possible number-execution-effects in step (10e).

13. A method according to claim 12, wherein said step (10e) "analysing the effects of this opcode-like execution of the number" further comprising the step of determining the number=opcode's mnemonic and its related source- and destination-registers by regarding all execution-effects of every initial condition.

14. A method according to claim 13, further comprising the step of valuating the effects of the execution of the number in reference to its basic needs, which means positive values for numbers, which cause no pain or increase the energy-register and negative values for numbers which cause above defined "pain" or "hunger".

15. A method according to claim 14, further comprising the step of combining numbers and executing these combinations and valuating the effects of the executions of these combinations.

16. A method according to claim 15, comprising the step of running several of these said programs as an extra process=task.

17 A method according to claim 15, comprising the step of using a network topology where on two or more of the networked computers is running one of these programs.

20 18. A method according to claim 14, further comprising the step of analysing a pregiven code by executing larger getting sequences of it instead of executing number-combinations, for to evaluate the effects these code-sequences or at least of the complete program.

21 19. A method according to claim 20 18, further comprising the step of improving the analysed code in the direction of a pregiven programming-aim.

18 20. A method according to claim 15, further comprising the step of implementing calls to operation-system routines which are offered in a table with entrance-address and source- and destination-registers.

19 21. A method according to claim 18 20, further comprising the step of using only calls and combinations for further use, which got a positive value from a valuation-function, which appraises the valuability of the call-combination in reference to reaching a pregiven programming-aim, not causing fatal exceptions, not overwriting exception-vectors or the program, avoiding to use forbidden registers or extensive writes to memory or large jumps, etc.